Name: Gaurav Patel
CUID: C19407183

Deep Learning Homework – 1

Task to Perform: -

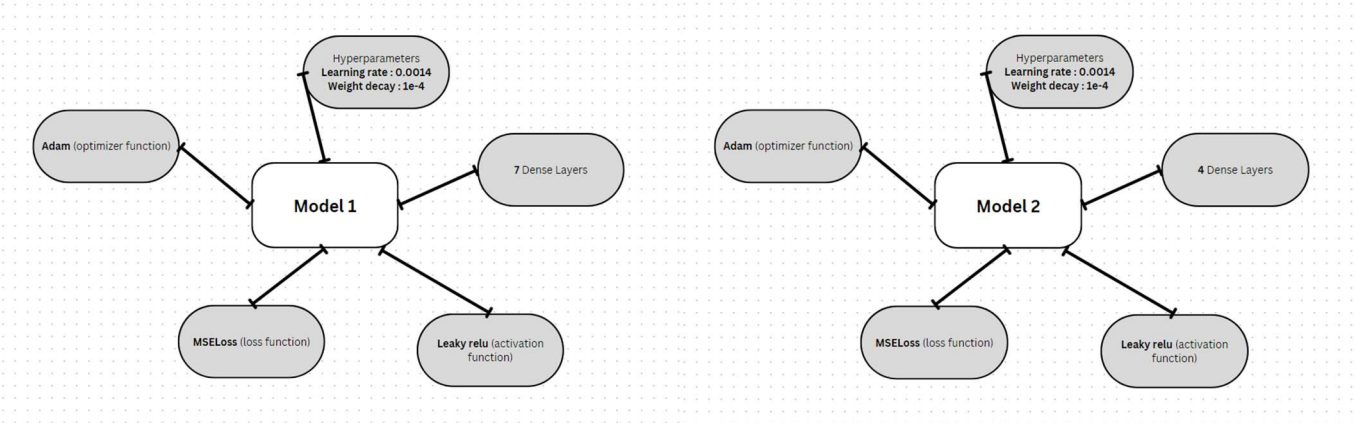
1.1) Simulate a function and Train the function using more than two DNN models with same number of parameters

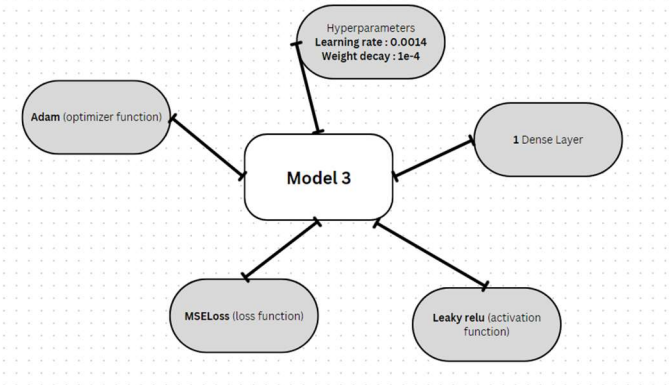
1.2) Optimization of the visualization process used MNIST dataset throughout the assignment. Also finding out what happens when the gradient is almost zero.

1.3) Generalization by fitting random labels by the network, comparison of the parameters vs generalization, and analysis of flatness vs generalization.

1.1) Deep vs Shallow

In the assignment, three deep neural network (DNN) models were developed to approximate the functions sin(5πx)/(5πx) and sgn(sin(5πx)). There were variations in the number of dense layers and parameters across all models. The regularization method called "weight decay" was used on all three models to avoid overfitting. The training procedure was programmed to terminate early if the model converged, which is defined as a virtually zero loss that stops decreasing, after a maximum of 30,000 epochs. The third DNN model was specifically configured as follows:

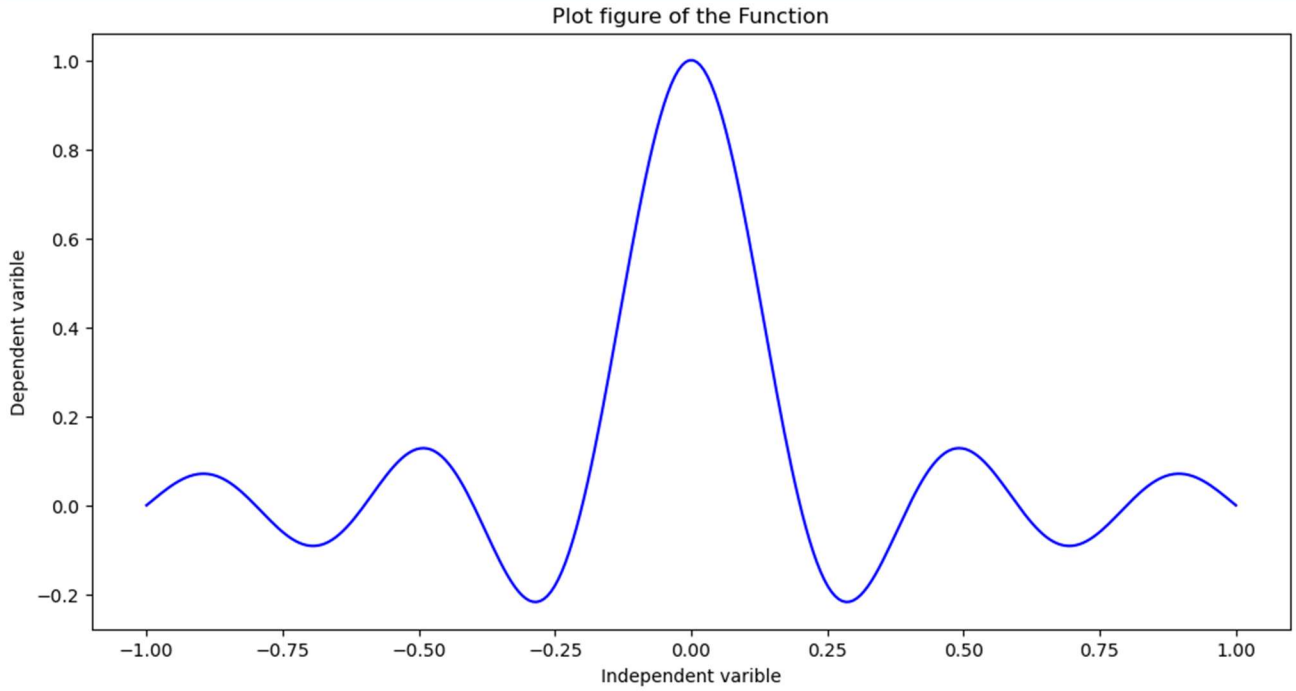




Total Parameters in each Models are around the same i.e. between 571 to 573 parameters.

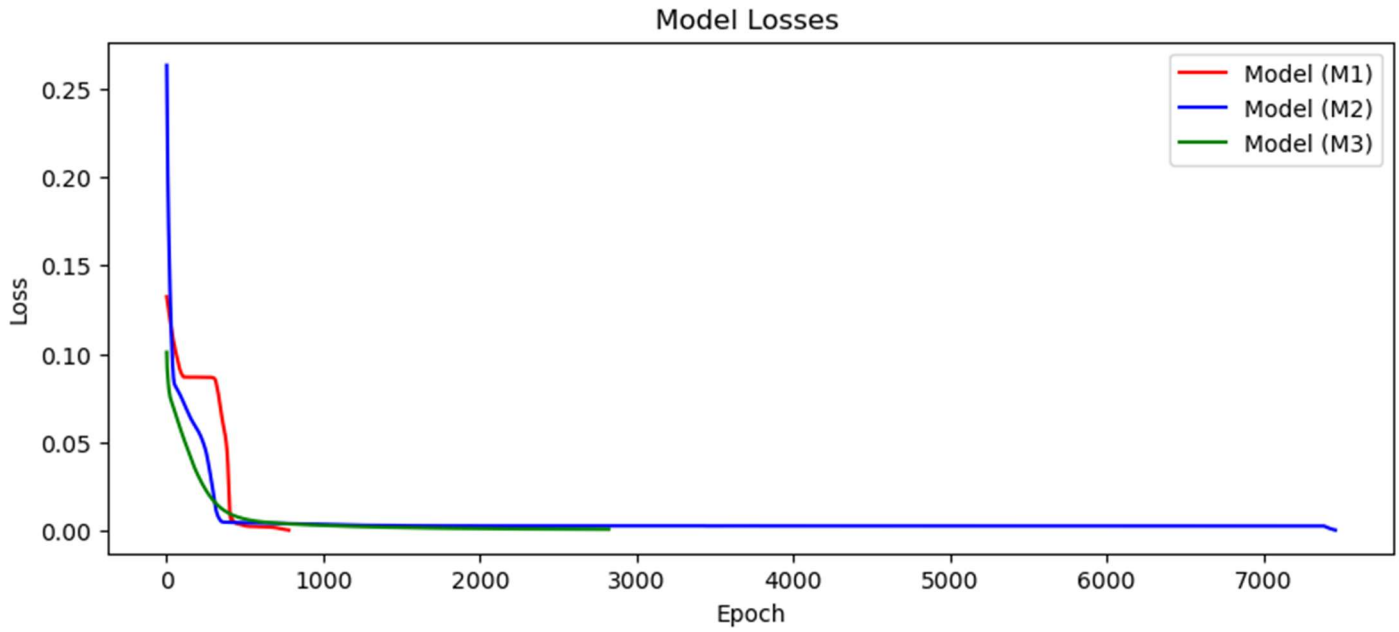
Function 1) sin(5πx)/(5πx)

Following the plot visualization of the function :

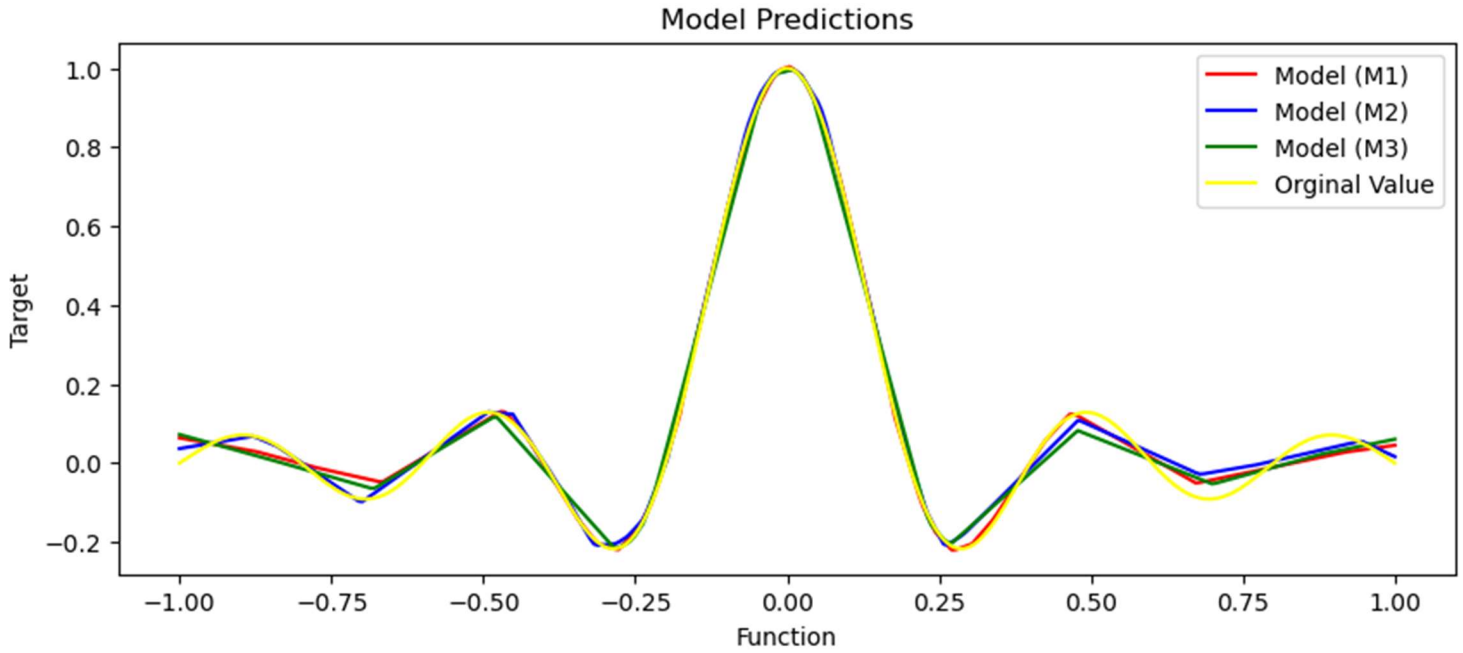


The models were converged at different number of epoch as mentioned below where Model 1 for (**epoch = 780**), for model 2 (**epoch = 7459**) and for model 3 (**epoch = 2422**).

The below graph shows the Model loss i.e. relationship between epoch and learning rate.



The below graph shows the Model prediction i.e. relationship between target and function.

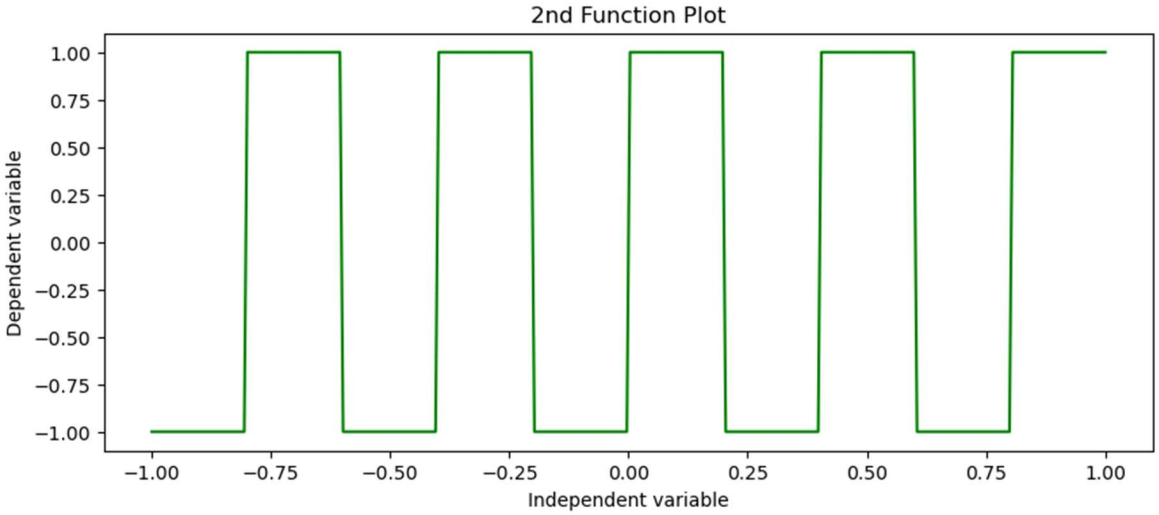


Observations:

1) Convergence Speed: With additional layers, Models 1 (M1) and 2 (M2) exhibit faster convergence than Model 3 (M3), which has only one layer. This is illustrated in Figure 2, where all models eventually stabilize at similar levels, but M1 and M2 attain near-zero loss more quickly than M3.
2) Model Performance: When it comes to forecasting the target function, all three of the models perform similarly, as seen in Figure 2. The models exhibit minimal departure from the ground truth, even with variances in architecture, suggesting that learning is effective across the board.
3) Effect of Layer Depth: Models 1 and 2's extra layers aid in accelerating convergence, which is advantageous when achieving faster training. Deeper models might, however, raise the possibility of overfitting, which is important to understand, particularly when working with smaller datasets. Furthermore, training deeper models may require more time and, if done improperly, may result in less-than-ideal convergence.

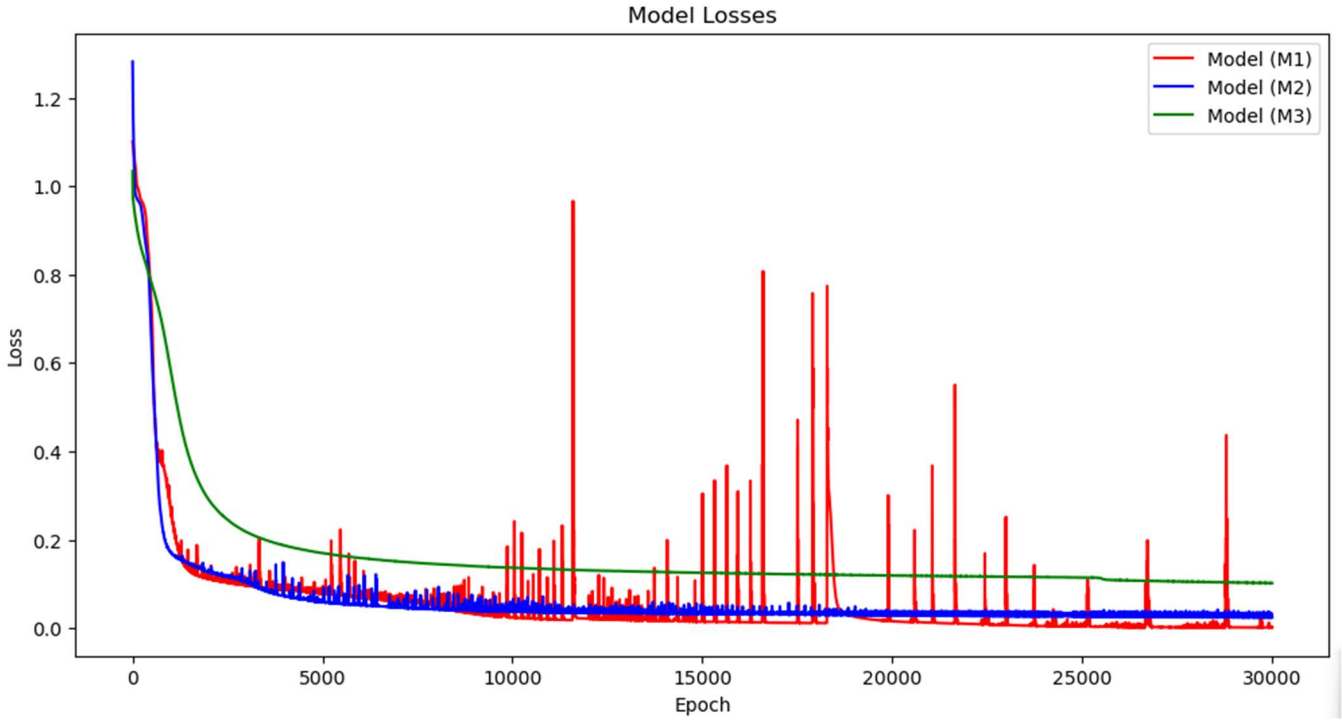
Function 2 : $\text{sgn}(\sin(5\pi x))$

Following the plot visualization of the function :

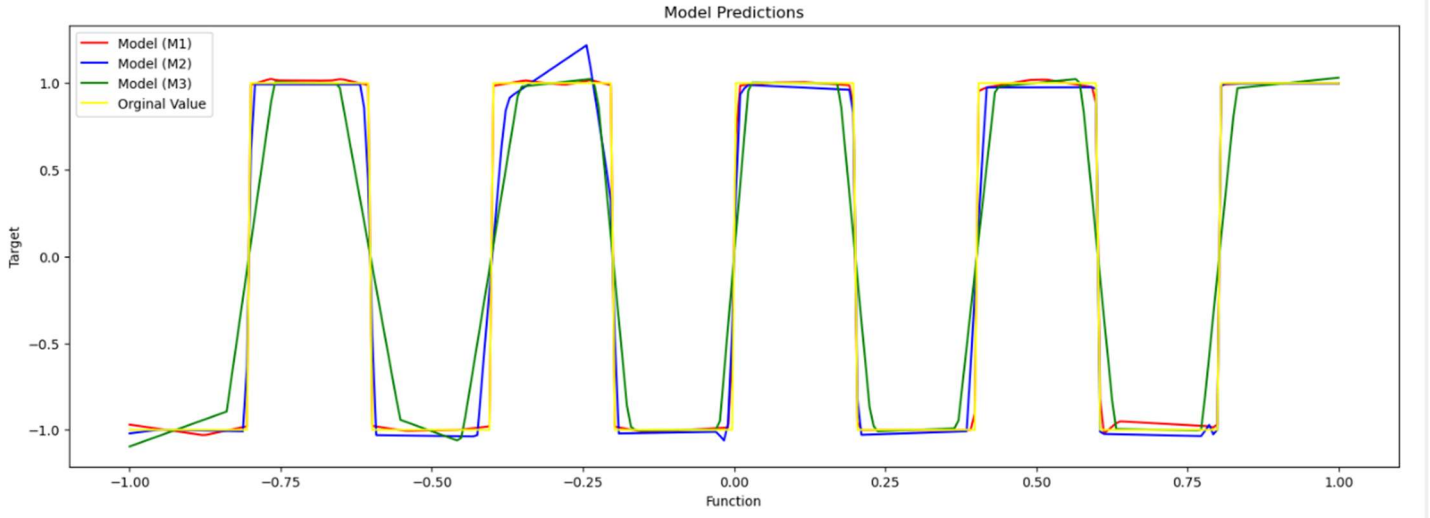


The models were converged at different number of epoch as mentioned below where Model 1 for (**epoch = 8146**), for model 2 (**epoch = 25000**) and for model 3 (**epoch = 25000**).

The below graph shows the Model loss i.e. relationship between epoch and learning rate.



The below graph shows the Model prediction i.e. relationship between target and function.



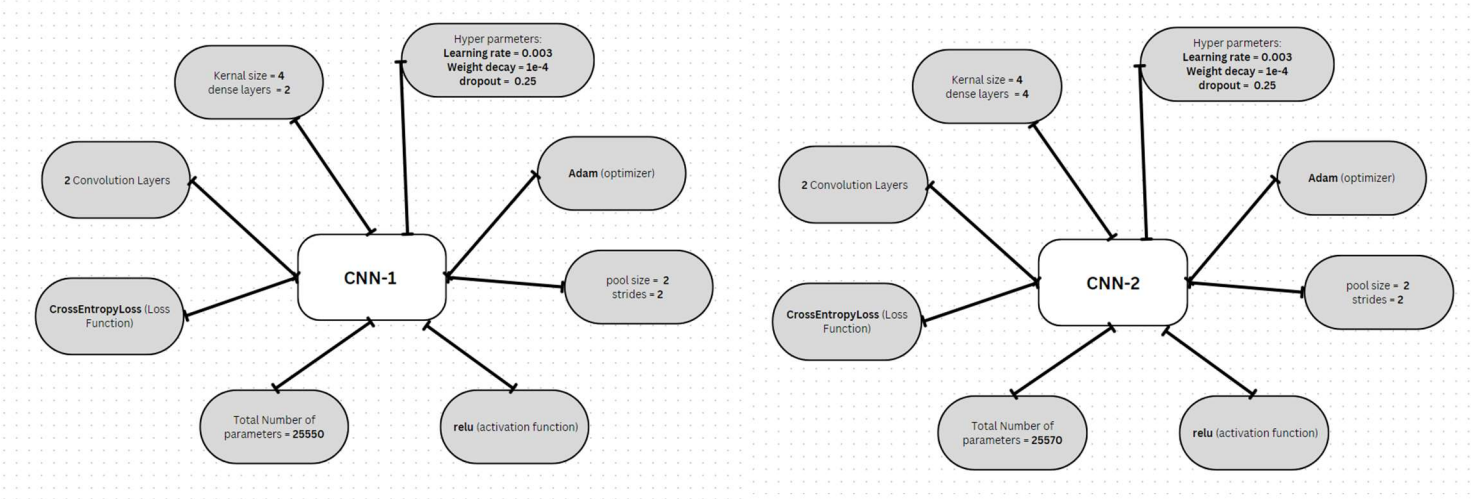
Observations:

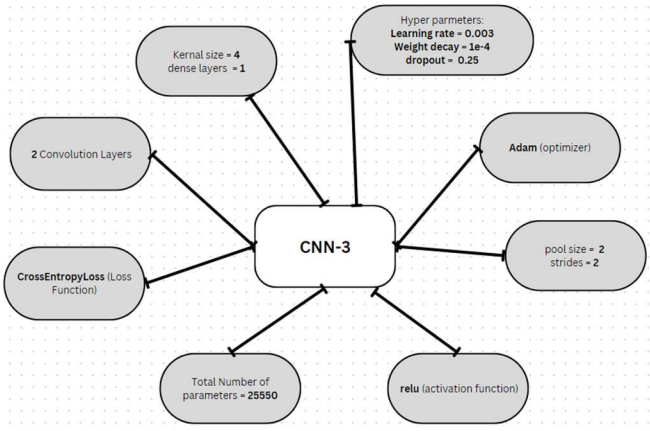
1) Convergence: As shown in Figure, Model 1 (M1) converged satisfactorily about epoch 8146, at which point its loss stabilized and remained modest throughout time. Nevertheless, even at the maximum number of epochs (25,000), Models 2 (M2) and 3 (M3) exhibit oscillation and fluctuation, suggesting that they did not fully converge.

2) Prediction Performance: M2 and M3's prediction accuracy is comparable to that of M1, even though they are unable to obtain convergence. With little variance throughout the predicted function, Figure 6 demonstrates that all three models (M1 in red, M2 in blue, and M3 in green) offer quite good approximations of the ground truth (yellow line).
3) Impact of Model Complexity: Even on a more complex function like this one, the model with the most layers (M1) converged more quickly, demonstrating the sometimes-beneficial nature of having more layers. It is crucial to remember that, depending on the data and issue, deeper models like M1 could be more prone to overfitting or require longer training cycles.

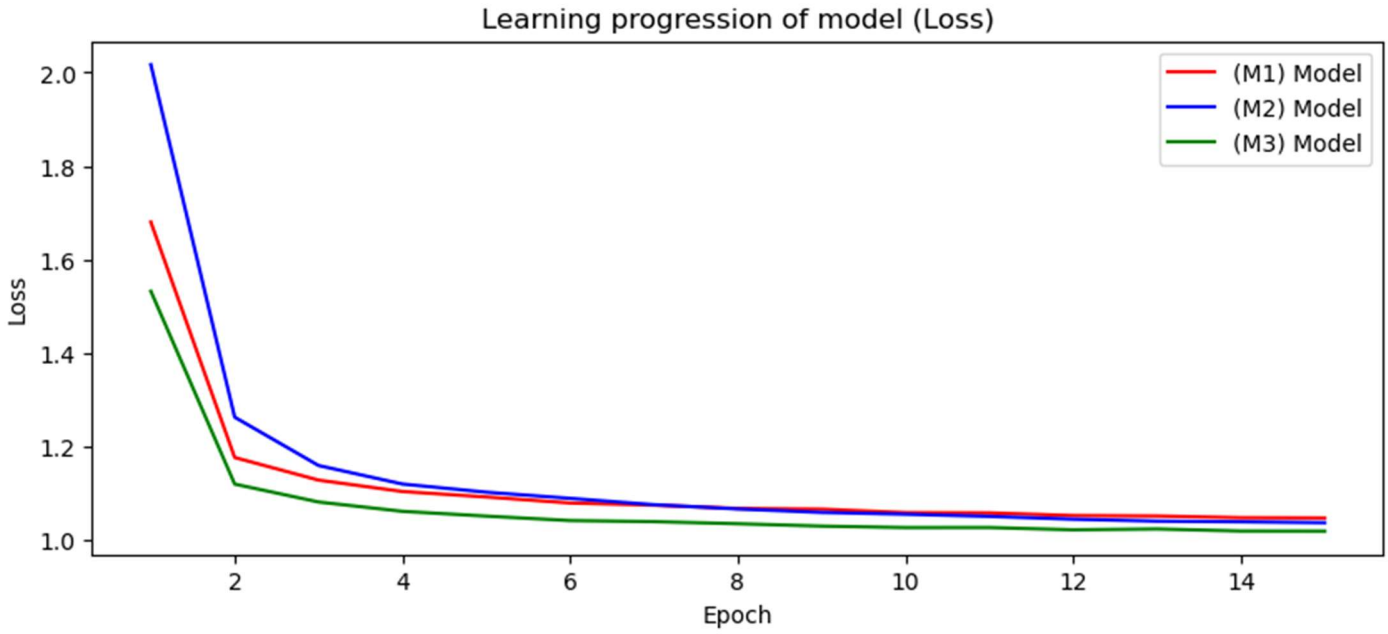
MNIST dataset (train on actual task):

Using the MNIST dataset, we created three Convolutional Neural Network (CNN) models for this project, each with a different layer structure but a comparable amount of parameters. The following is a summary of each model's particular details:

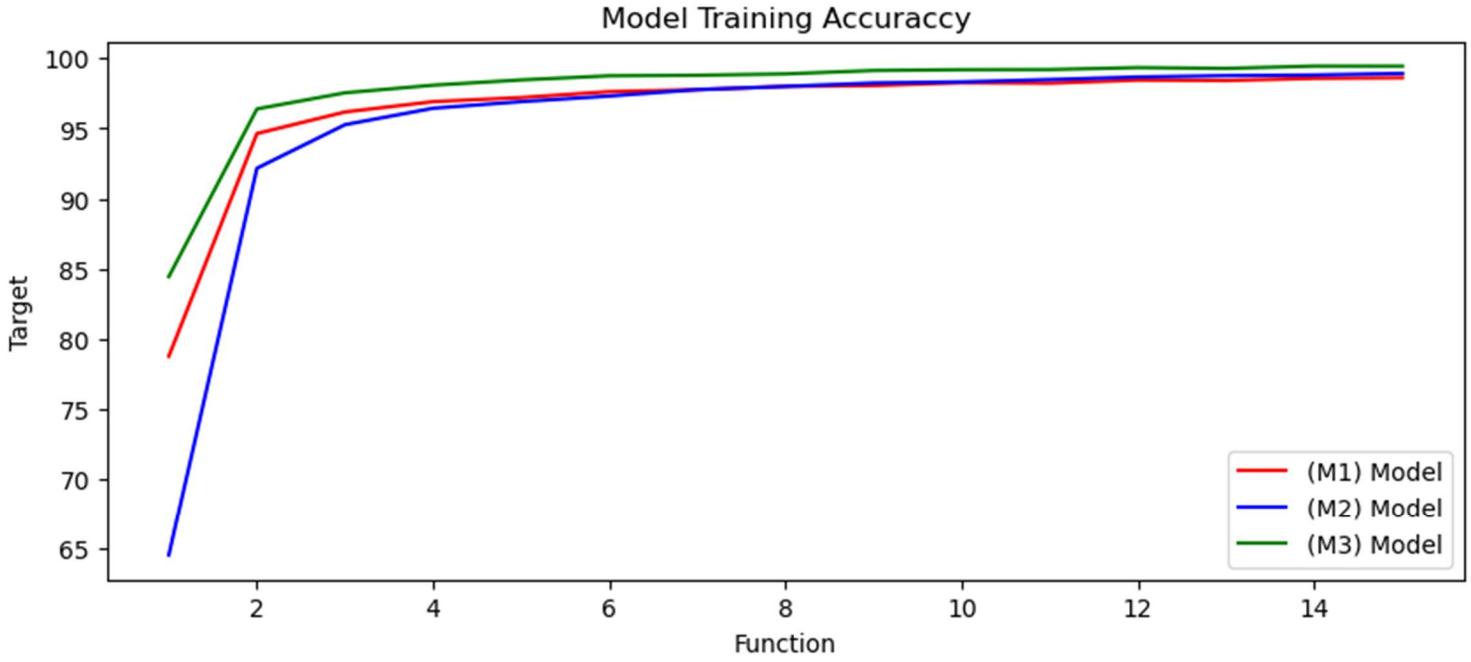




The below graph shows the learning progression of the model i.e. Loss



The below graph shows the Model Training Accuracy




```
CNN1 Testing Accuracy: 99.04 %
CNN2 Testing Accuracy: 98.69 %
CNN3 Testing Accuracy: 99.01 %
```

Observation:

1) Model Loss: Out of the three models, Model 3 (M3, green) exhibits the lowest loss values throughout the training process, suggesting superior optimization. Both Model 2 (M2, blue) and Model 1 (M1, red) show consistent drops in loss but converge to marginally higher levels. The final loss for Model 3 is 0.0242, whereas the losses for Model 1 and Model 2 are 0.0744 and 0.0880, respectively.

2) Model Accuracy: Model 3 slightly outperforms the other two, with a training accuracy of 99.01% and the highest test accuracy of 99.01%. All three models demonstrate great accuracy. Models 1 and 2 obtain test accuracy of 99.04% and 98.69%, respectively.

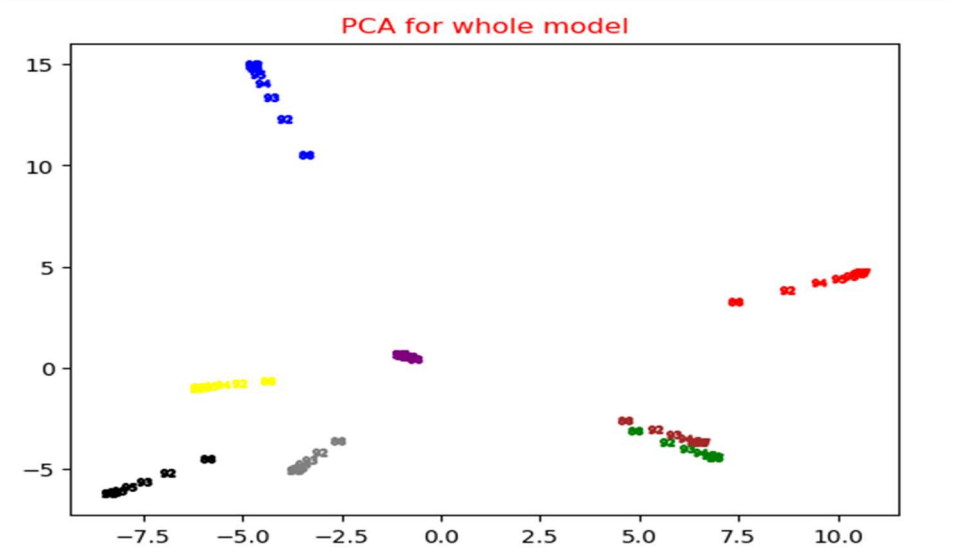
3) Performance Comparison: Model 3 performs better than Models 1 and 2 in terms of training loss and accuracy even though it only has one dense layer. This implies that, despite having a simpler configuration, Model 3's architecture is more effective for this dataset.

1.2 Optimization

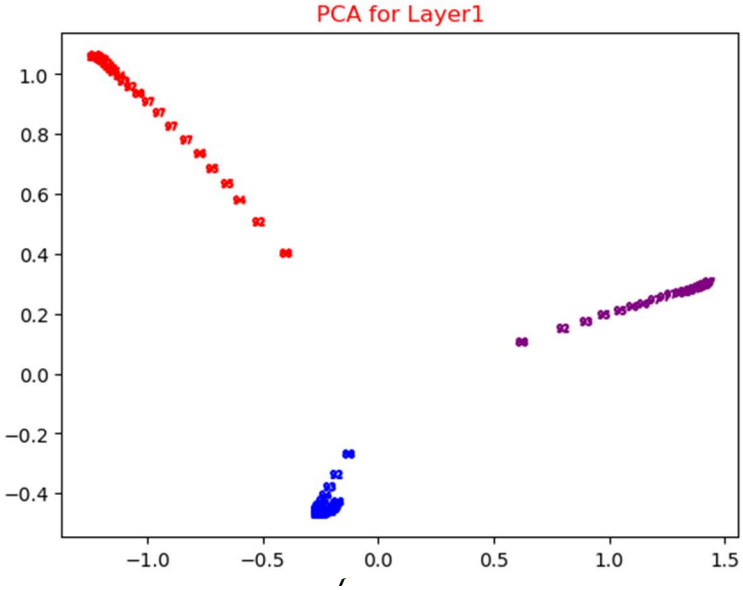
Visualize the optimization process:

A Deep Neural Network (DNN) with a single dense layer and the rectified linear unit (ReLU) activation function is the model used for this portion of the assignment. There are 418,060 parameters in all. The Cross-Entropy Loss function and the Adam optimizer, set up with a weight decay of 1e-4 and a learning rate of 0.0002, were used in the training process. The model underwent 30 epochs of training, with the weights recorded at the end of each epoch. The training process was then repeated eight times.

The below graph shows the PCA visualization of the whole Model



The below graph shows PCA for Layer 1



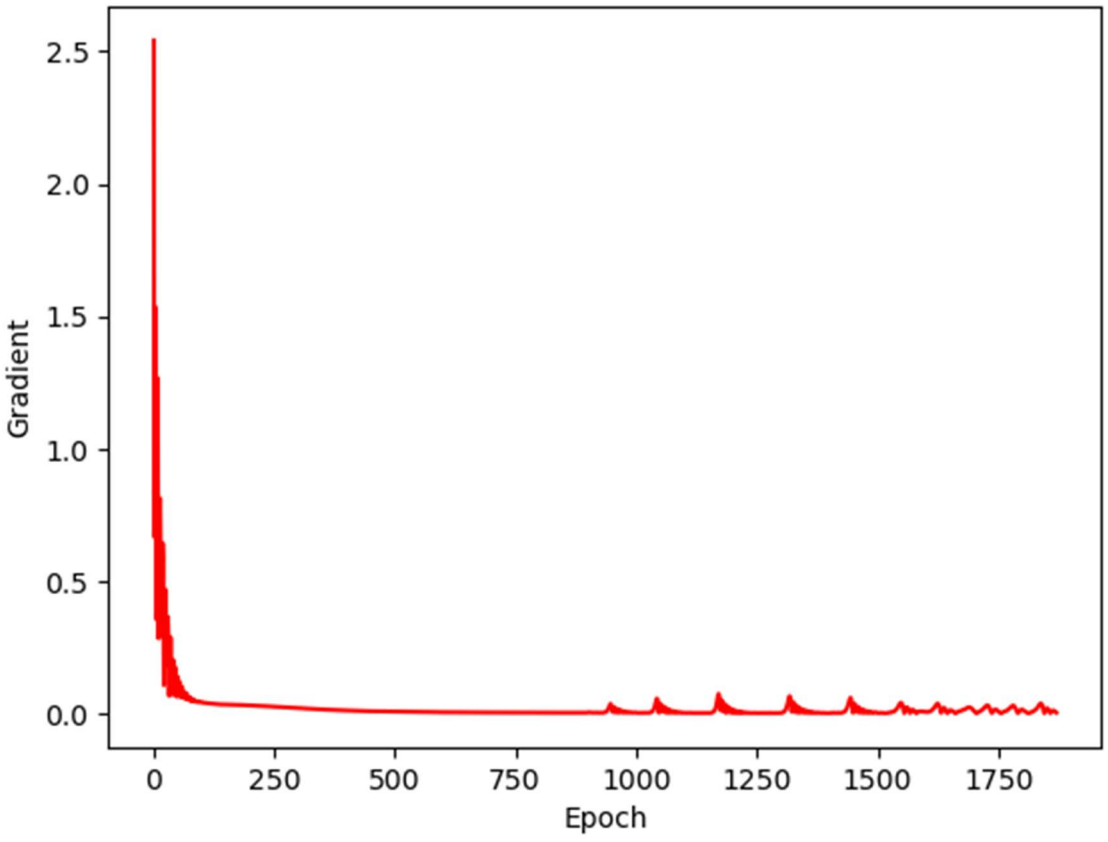
Observations:

1) Principal Component Analysis (PCA) was used to reduce the dimensionality of the weights from 418,060 to two dimensions using the weights derived from all training runs. After that, a plot of the PCA-transformed weights was created to show the weight distribution and the optimization process throughout the several training runs.
2) Different training runs' worth of weight values are clearly clustered together in the first graph. A distinct epoch appears to be represented by each color, illustrating how the model's weights change across training. Clusters that are somewhat separated from one another suggest that the model optimizes differently for each training run.
3) The PCA findings for Layer 1 are displayed in the second graph, which sheds light on how the weights in this layer changed over training. It is evident that there is clustering with definite separations, indicating that the weights in the first layer followed a particular trajectory or pattern over time.

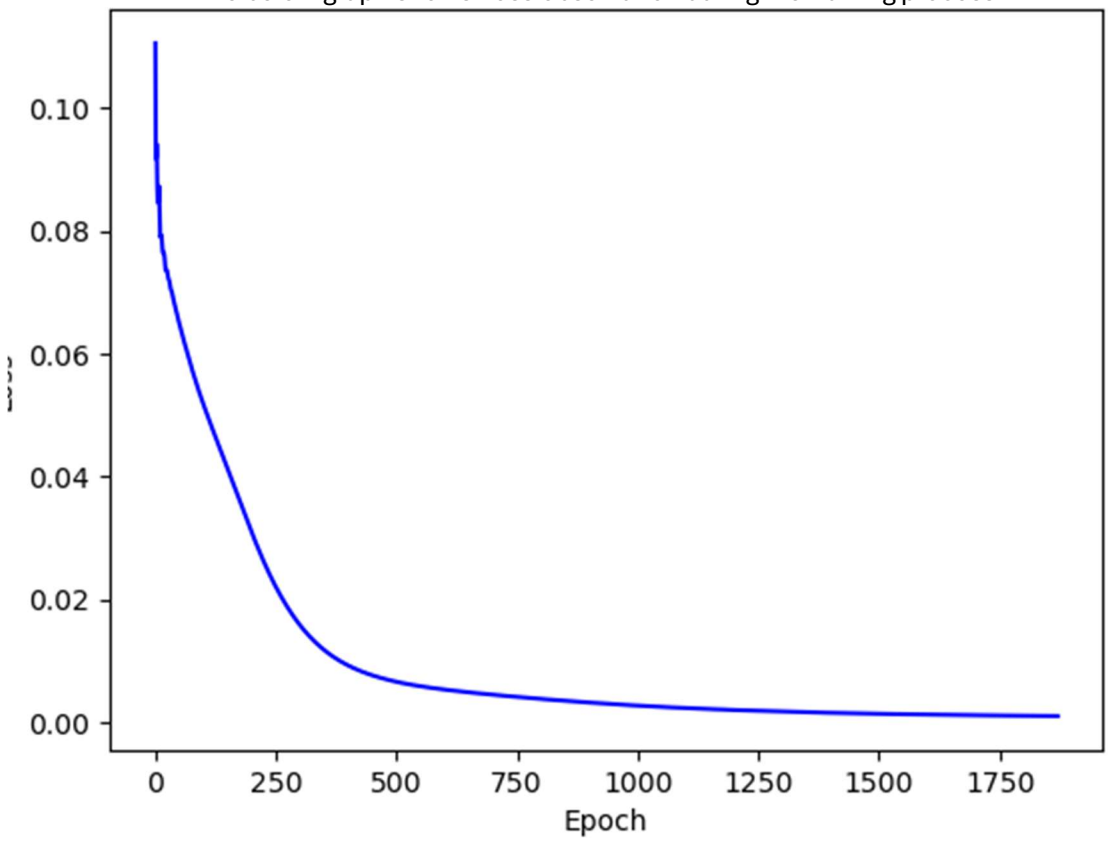
Gradient Norm during the visualization process (using the function : sin(5πx)/(5πx)):

A Deep Neural Network (DNN) with a single dense layer and the rectified linear unit (ReLU) activation function is the model employed in this portion of the assignment. There are 1,501 parameters in the model altogether. The Adam optimizer was utilized for optimization, and the Mean Squared Error (MSE) loss function was implemented during training. Hyperparameters used in the optimization procedure included a weight decay of 1e-4 and a learning rate of 1e-3. The model obtained convergence with a final MSE loss value of 0.0009992709 after 1,800 epochs of training. Following convergence, the gradient norm—which gauges the loss's rate of change in relation to the model's parameters—was also kept an eye on.

The below graph shows the Gradient norm observation during the training process.



The below graph shows Loss observation during the training process.



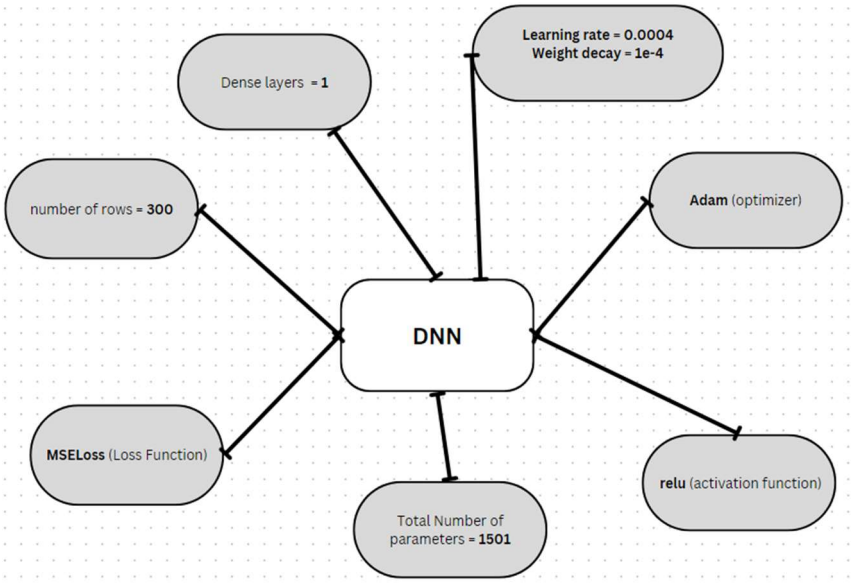
Observations:

1) The graph illustrates how the loss decreased during the training epochs. The loss initially decreases sharply, especially up to about 400 epochs, indicating that the model is performing better and faster. The loss then starts to decline again, albeit more slowly. Eventually, it reaches a plateau, suggesting that the model has probably found convergence. By the completion of the training process, the model had reached a high level of optimization, as evidenced by the final MSE loss value of 0.0009992709.
2) The gradient norm quantifies the pace at which the loss varies in relation to the parameters of the model. The graph indicates that there is a notable initial decline in the gradient norm, particularly in the first 200 epochs, which is consistent with large updates in the model's parameters. Following this abrupt decrease, the gradient norm stabilizes and stays low, with only slight variations around the 900th epoch. This suggests that the model's parameters have found an ideal range for minimizing the loss.

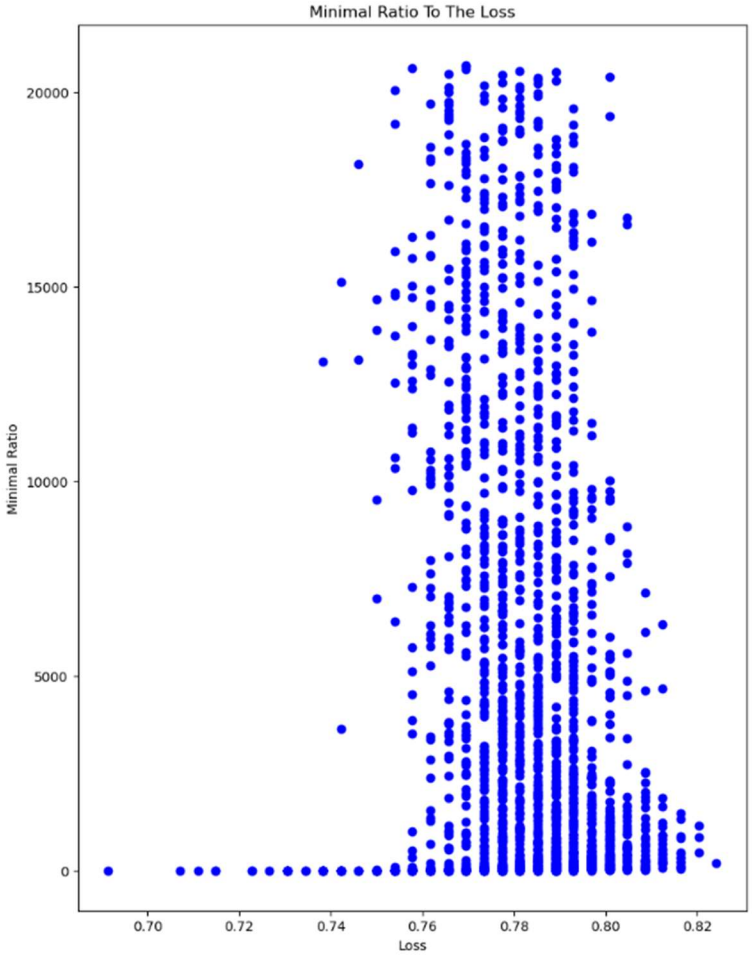
Most of the learning occurs during the first few hundred epochs, with learning becoming less effective beyond that. This analysis aids in selecting the ideal number of epochs for upcoming training.

What Happened When Gradient is Almost Zero:

In this section of the assignment, a Deep Neural Network (DNN) model and function were utilized to observe when the gradient approaches zero and to compute the minimum ratio. After training the model for 100 epochs, it was observed that the loss did not reach zero. **Figure 13** illustrates the graph of the loss versus the minimum ratio.



The below graph shows the relationships between Loss and Minimal ratio

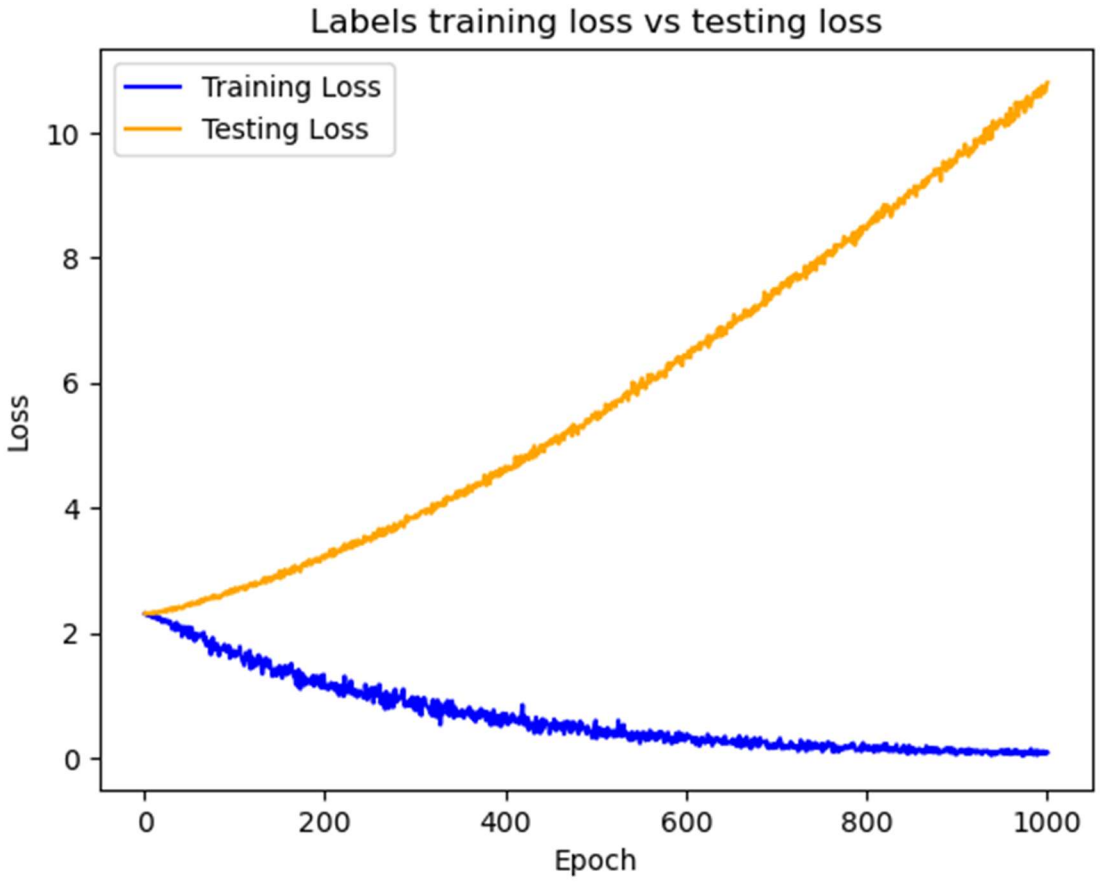


1.3) Generalization:

Can network fit random labels?

In this study, we use the MNIST dataset, whose labels have been randomly selected, to train a Deep Neural Network (DNN) model. The objective is to watch the model's behavior as it learns and evaluate its results against test data that consists of correctly identified images. The model uses the ReLU activation function and has a single dense layer with 397,510 parameters. During training, the Adam optimizer is employed with a learning rate of 0.0011 and the Cross Entropy Loss function.

The below graph shows Random Fit labels vs Test Loss.



Observations:

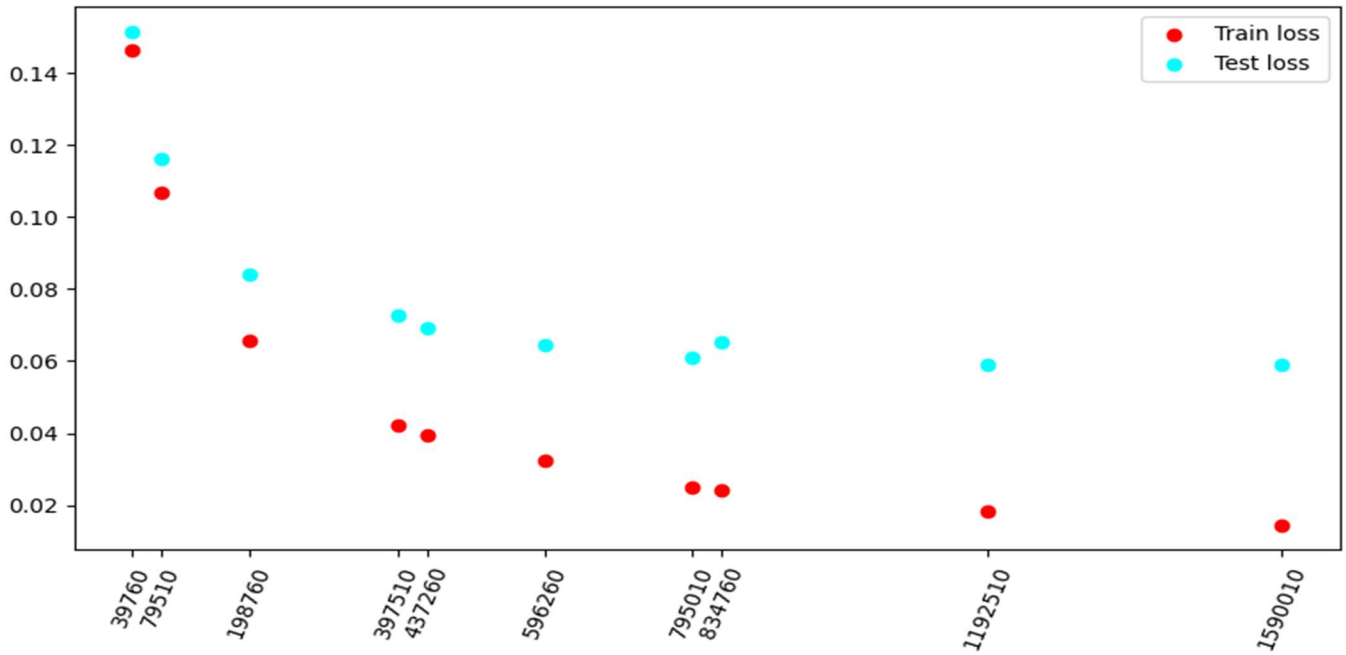
1) The training loss is shown by the blue line, which progressively drops as the model gains knowledge from the randomized labels. The training loss approaches zero at the conclusion of 1,000 epochs, suggesting that the model is overfitting to the noise in the training set. For the training set, the model minimizes the loss in spite of the inaccurate connections between the images and labels.
2) The testing loss is represented by the orange line, and it rises steadily over the course of the epochs in sharp contrast to the training loss. As the training goes on, the loss increases dramatically since the model cannot generalize to the correctly labeled test set because it was trained on inaccurate labels.

The testing loss increasing with time amply illustrates how poorly the model generalizes. Therefore, using randomized labels in training is not a good way to learn or make accurate predictions.

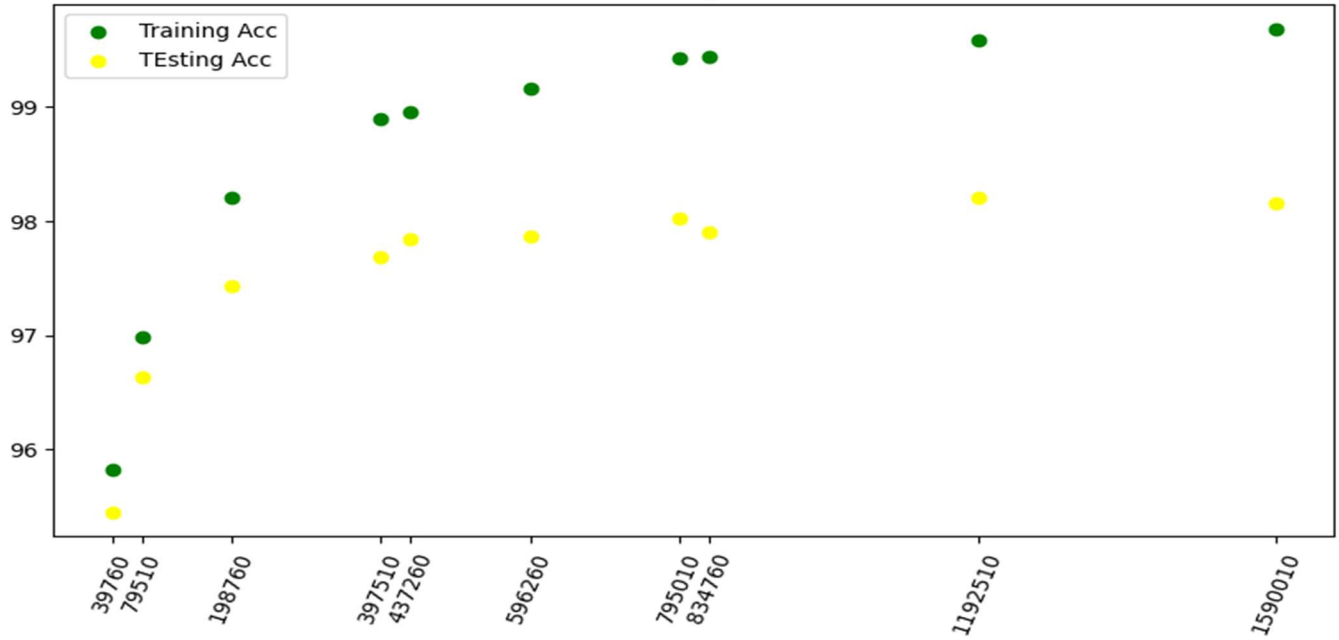
Number of parameters v.s. Generalization:

Ten Convolutional Neural Network (CNN) models with comparable architectures were constructed for this assignment portion; however, the models' dense layer values differed. 39,760 to 15,90010 features were included in these models. Two convolutional layers with a kernel size of four made up each model. Each model also used the ReLU activation function and had two dense layers. The total amount of parameters across the models was 25,550. The training procedure used the Adam optimizer with a learning rate of 0.0011 and Cross Entropy Loss as the loss function. Additionally, a 0.25 dropout rate was used to avoid overfitting.

The below graph shows Train Test Loss vs Parameters



The below graph shows Accuracy comparisons



Observations:

1) Both the test loss and the training loss tend to decrease as the number of parameters grows. When a certain number of parameters is achieved, the test loss starts to fluctuate less between models and achieves a plateau considerably earlier than the training loss. According to this, overfitting is probably to blame for the models' poor performance on test data after a certain point, even while their training loss is improved by the models with more parameters.
2) Likewise, an increase in the number of parameters leads to a continuous improvement in training accuracy. However, especially for models with more parameters, the test accuracy peaks early and there is a discernible difference between the training and testing accuracy. Given that the models perform well on the training data but less well on the test data that has not been seen, this gap suggests that the models are probably overfitting.

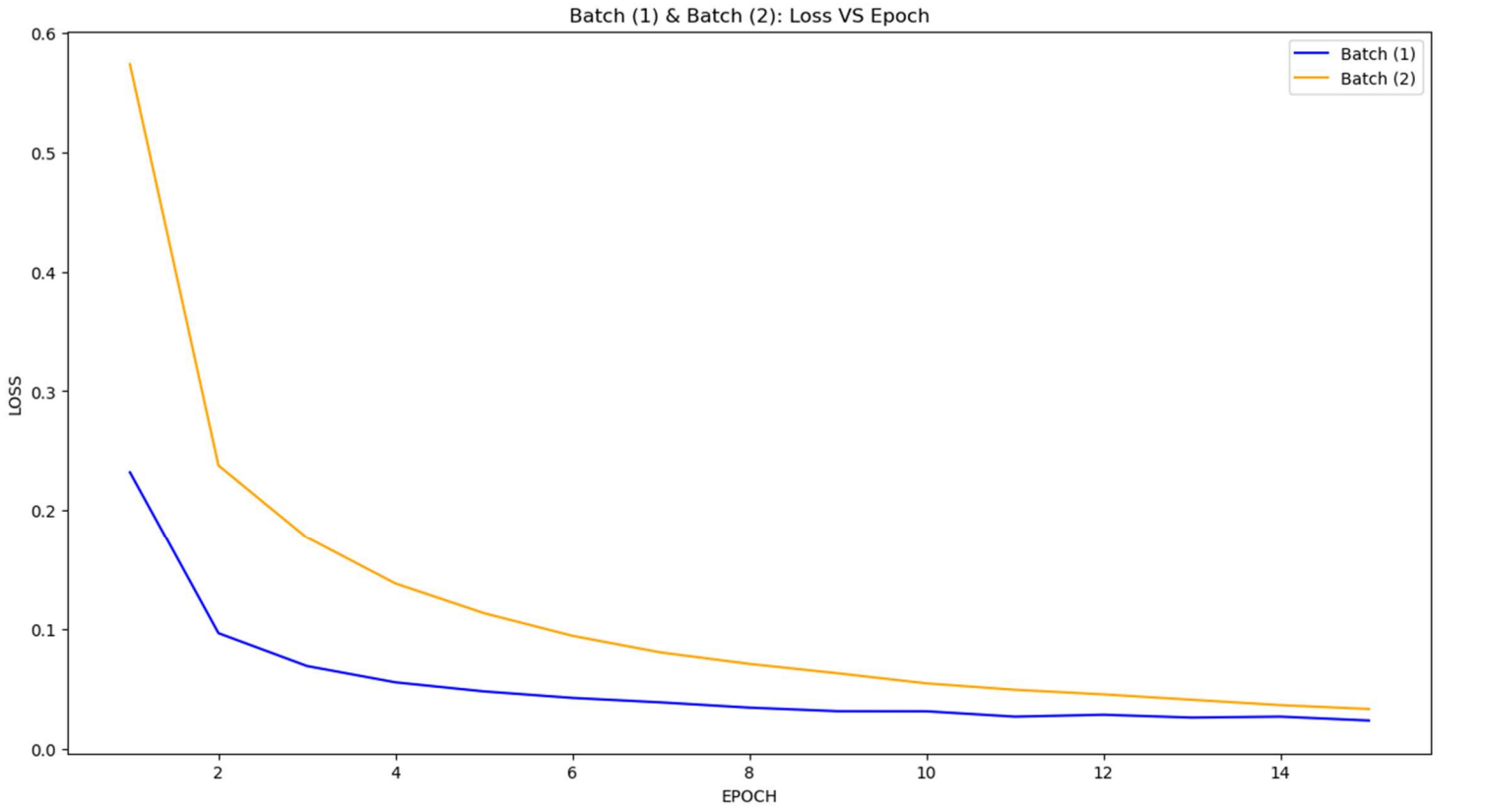
When a model grows overly complicated and begins to learn from the training data rather than generalizing to new, untried data, this is known as overfitting. As a result, there is a discrepancy between training and test accuracy even though the model performs well on the training set.

Flatness vs Generalization: -

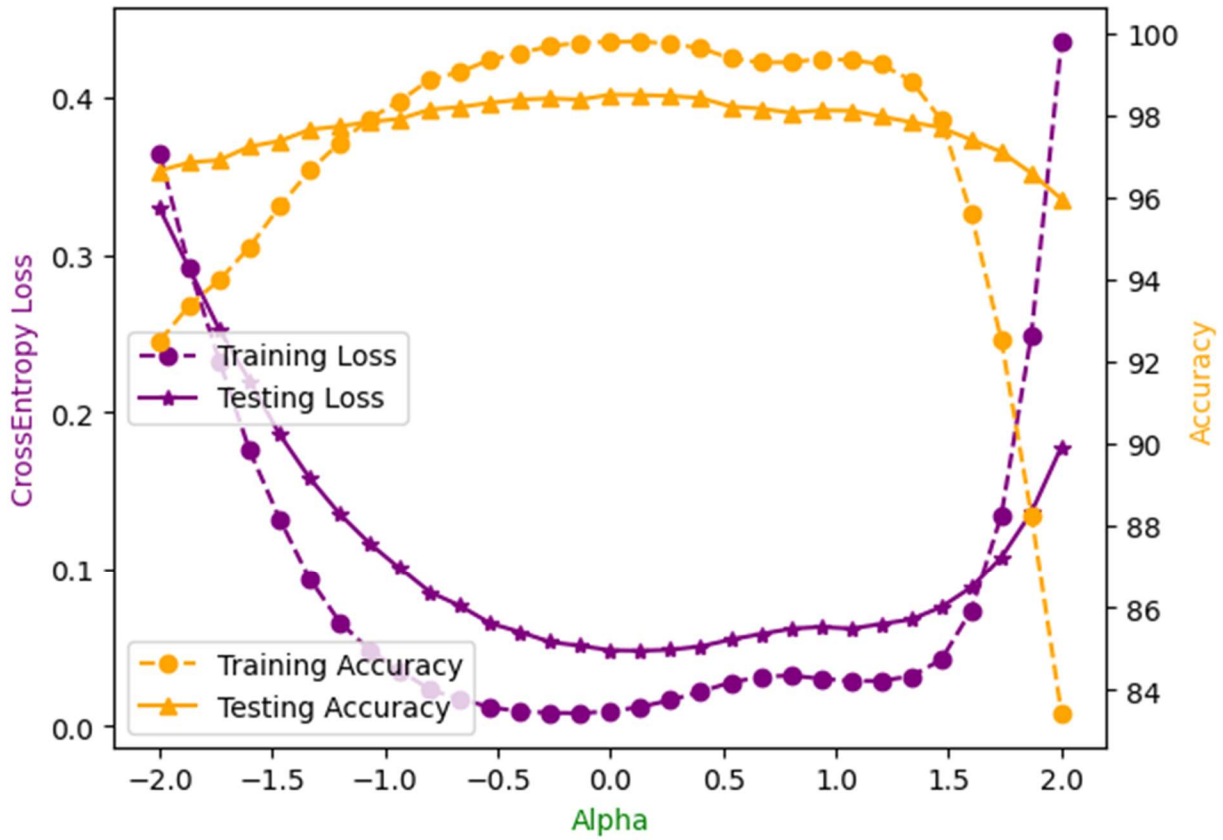
Part 1:

Two deep neural network (DNN) models with the same topologies but different batch sizes for training have to be developed for this project. Both models were trained on the MNIST dataset; the first will employ a batch size of 64, and the second, a batch size of 1000. With the ReLU activation function, each model will have a single dense layer. There will be 397,510 parameters in the network overall. The Adam optimizer will be used to optimize the training, and CrossEntropyLoss will be used as the loss function. With a weight decay of 0.0001 and a learning rate of 0.0015, the training will last up to 15 epochs.
The formula used to calculate ϑ :- $\vartheta = (1 - \alpha) * batch1_{param} + \alpha * batch2_{param.}$

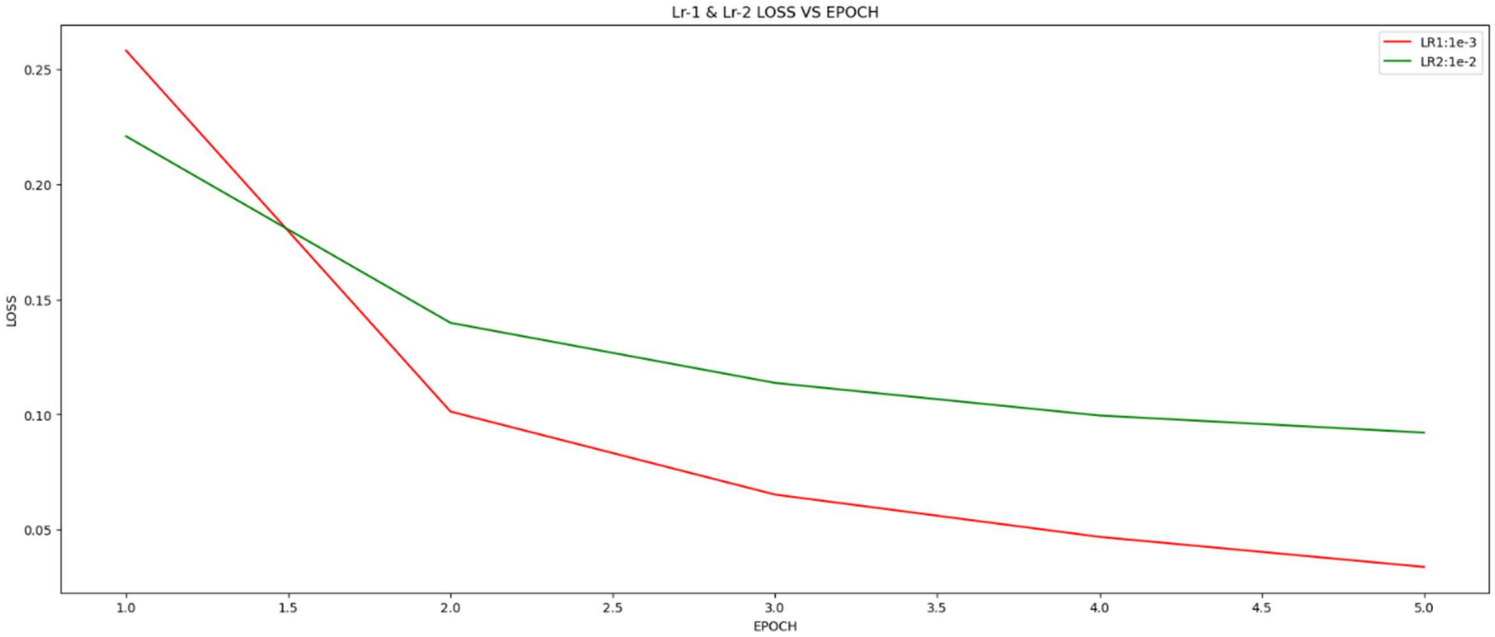
The below graph shows the comparison of Batch 1 and Batch 2 in terms of Loss vs Epoch.



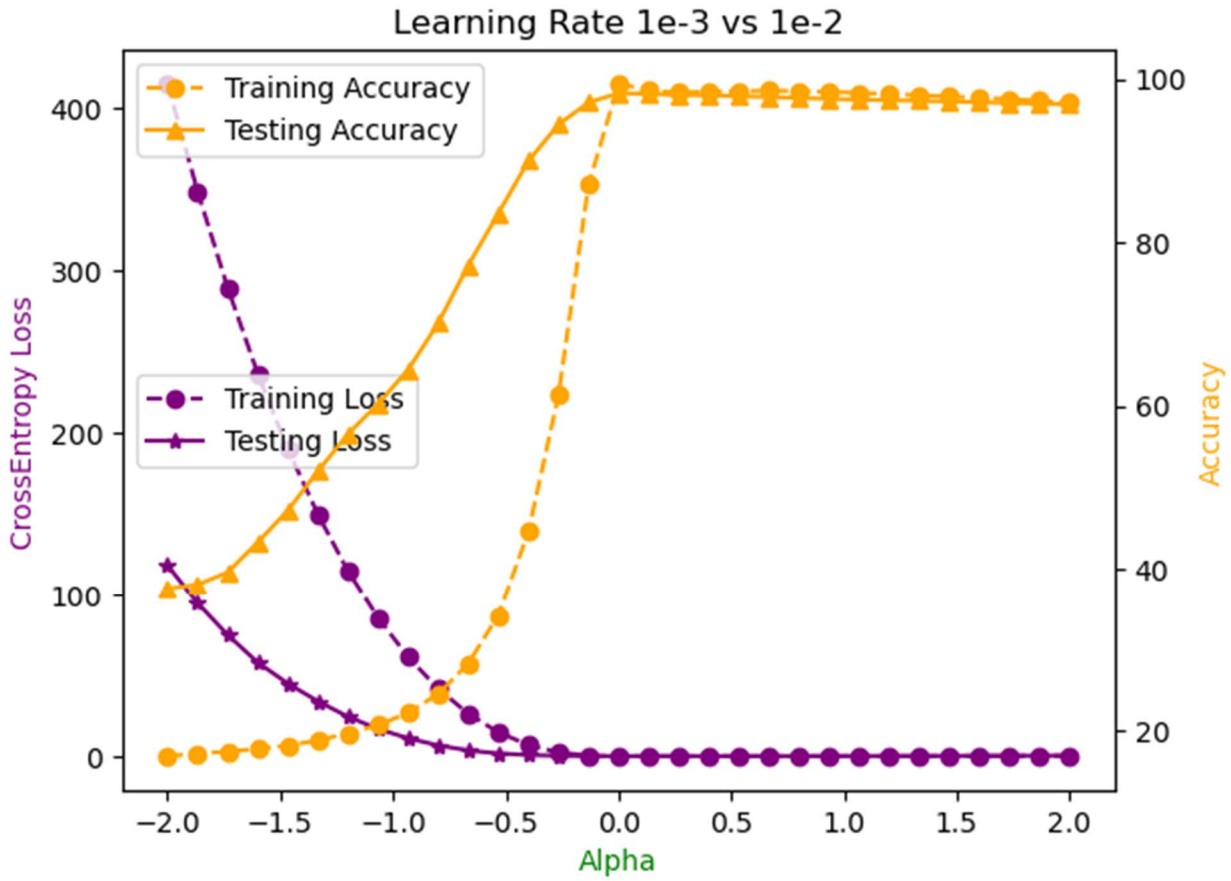
The below graph shows loss and accuracy were observed per alpha.



The below graph shows model with different learning rates.



The below graph shows Loss and Accuracy per alpha values.



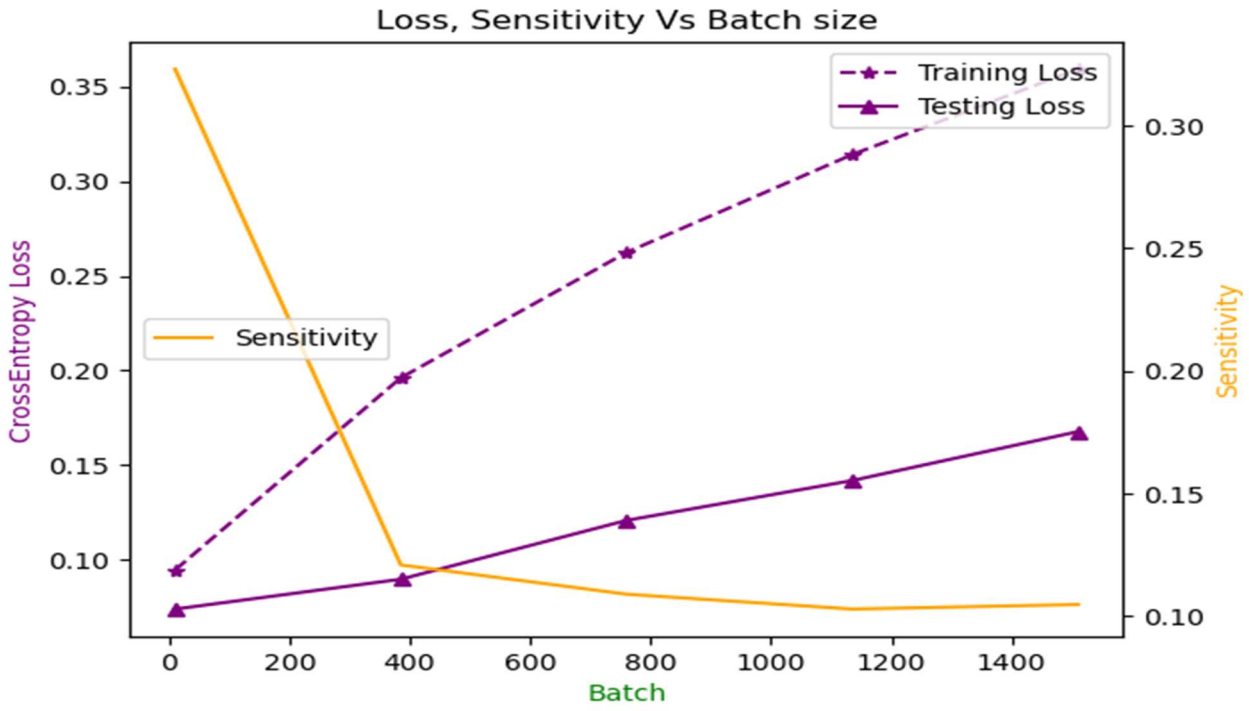
Observations:

1) A smaller batch size (64) results in faster convergence and a lower final loss compared to a larger batch size (1000) in the first graph, which displays models trained with varying batch sizes. Greater batch sizes have slower convergence and a higher initial loss.
2) The second graph shows that accuracy and loss decrease as alpha gets closer to 0.0. But for alpha values between 1.5 and 2.0, there is a sharp reduction in performance that is followed by a temporary improvement. This is probably because to overfitting or instability at higher alpha values.

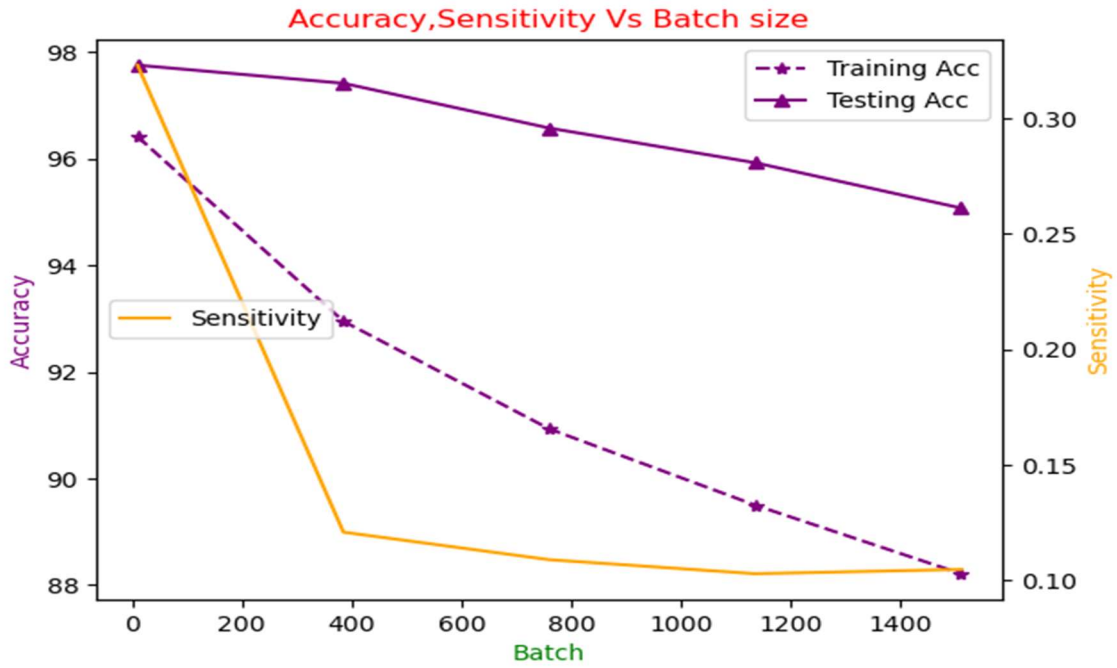
Part 2:

This section involves the training of five DNN models with varying batch sizes (10, 385, 760, 1135, and 1510) using the MNIST dataset. With one dense layer with ReLU activation, 397,510 total parameters, cross entropy loss, the Adam optimizer, and a training restriction of five epochs, all models have the same architecture. A weight decay of 10^{-4} and a learning rate of 10^{-3} are two of the hyperparameters that were used. Sensitivities, loss, and accuracy are monitored during training to investigate the impact of batch size on model performance.

The below graph shows Loss, Sensitivity vs different batch size.



The below graph shows Accuracy, Sensitivity vs Batch size



Observations:

1) The displayed graphs make it evident that the model's performance tends to decline over the same number of epochs as the batch size grows. The first graph shows that as batch size increases, both training and testing loss rise, suggesting that models trained with smaller batch sizes are better at minimizing loss. This implies that the model can learn more quickly and effectively with a smaller batch size.

2) Conversely, a sharp dip in the sensitivity curve indicates that the model's sensitivity reduces significantly as batch size grows. This implies that the model gets less sensitive to changes in the input data as the batch size increases.
3) The accuracy of training and testing decreases steadily as batch size increases in the second graph. Smaller batches also yield higher accuracy, which lends credence to the notion that smaller batches facilitate greater learning. Additionally, sensitivity has a declining trend, supporting the finding that models with greater batch sizes are less sensitive.